Variational Harmonic Maps for Space Deformation

Mirela Ben-Chen Ofir Weber Craig Gotsman
Technion – Israel Institute of Technology

Abstract

A space deformation is a mapping from a source region to a target region within Euclidean space, which best satisfies some user-specified constraints. It can be used to deform shapes embedded in the ambient space and represented in various forms – polygon meshes, point clouds or volumetric data. For a space deformation method to be useful, it should possess some natural properties: e.g. detail preservation, smoothness and intuitive control. A harmonic map from a domain $\Omega \subset R^d$ to R^d is a mapping whose d components are harmonic functions. Harmonic mappings are smooth and regular, and if their components are coupled in some special way, the mapping can be detail-preserving, making it a natural choice for space deformation applications. The challenge is to find a harmonic mapping of the domain, which will satisfy constraints specified by the user, yet also be detail-preserving, and intuitive to control. We generate harmonic mappings as a linear combination of a set of harmonic basis functions, which have a closed-form expression when the source region boundary is piecewise linear. This is done by defining an energy functional of the mapping, and minimizing it within the linear span of these basis functions. The resulting mapping is harmonic, and a natural "As-Rigid-As-Possible" deformation of the source region. Unlike other space deformation methods, our approach does not require an explicit discretization of the domain. It is shown to be much more efficient, yet generate comparable deformations to state-of-the-art methods. We describe an optimization algorithm to minimize the deformation energy, which is robust, provably convergent, and easy to implement.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling
Keywords: Space deformation, harmonic maps, shape editing

1 Introduction

Space deformation methods deform the ambient space in which a shape is embedded, instead of explicitly deforming the shape itself. Such methods have become popular in recent years [Huang et al. 2006; Lipman et al. 2007b; Joshi et al. 2007, Lipman et al. 2008; Sumner et al. 2007; Botsch et al. 2007], for several reasons. First, they are more general than explicit deformation – space deformation can be applied to any shape representation, whether it is a polygonal mesh, a point cloud or volumetric data. Second, by deforming the ambient space, the computational complexity of the deformation is decoupled from the complexity of the shape, hence even extremely complex shapes can be deformed at interactive rates.

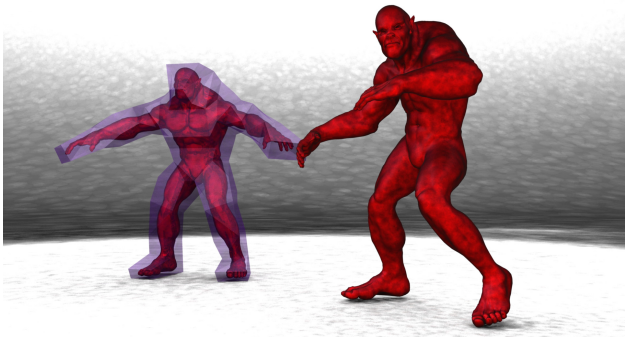


Figure 1: *The Beast model enclosed in its cage (left) and its deformation using a variational harmonic map (right)*

Some space deformation methods [Lipman et al. 2007b; Joshi et al. 2007; Lipman et al. 2008] are "cage-based". In these methods, a given "source cage" is manipulated by the user to create a "target cage". Then, based on the source and target cages, a mapping of the source cage is defined. If the mapping function has a closed-form expression, the deformation method becomes accurate and efficient. On the other hand, manipulating a cage is a tedious and time-consuming task. A more user-friendly and natural deformation method is direct manipulation – the user positions a small number of "control points" inside the domain, and manipulates them instead of the cage. Such methods [Huang et al. 2006; Sumner et al. 2007; Botsch et al. 2007] define the space deformation on a domain which is coarser than the input shape, and solve an optimization problem to find the parameters of the deformation, given the user's constraints. As this optimization problem is generally non-linear, the robustness and efficiency of these algorithms depend critically on the formulation of the deformation, and the optimization method used.

We propose to use harmonic maps of the source region as the underlying deformation model. Since harmonic functions are smooth and regular, they are used for a wide range of applications, from parameterization [Floater and Hormann 2005] and remeshing [Dong et al. 2005] to space deformations [Joshi et al. 2007; Lipman et al. 2008]. We generate harmonic maps on the domain as a linear combination of harmonic basis functions. In the special case that the domain is a polyhedron, these basis functions, and their first and second derivatives, will have closed-form expressions, as will the harmonic maps. Using these expressions, we allow the user to place position and orientation constraints at arbitrary locations inside the domain and define an energy functional which depends also on these constraints. By defining additional "rigidity lines" in a semi-automatic way, the resulting deformation is a natural "As-Rigid-As-Possible" deformation of the shape, respecting the specified constraints. It is worth noting that our harmonic basis functions are a variant of the "Green coordinates" of Lipman et al. [2008] (and Weber et al. [2009]), however, we give simpler expressions for them, and also provide their first and second derivatives.

1.1 Contribution

Our main contribution is a robust and very efficient space deformation method, which provides some advantages over existing methods. First, the user manipulates a set of position and orientation constraints, instead of directly manipulating the "source cage", hence our method is more intuitive and easy to control. Second, we have a closed-form expression - a linear combination of basis functions - for the deformation of a continuous domain, thus do not require a voxelization of the input domain, as some other methods do. And finally, since we have closed-form expressions also for the gradients of the deformation, our optimization procedure may be based on an alternating least-squares "local/global" algorithm, which, until now, was applicable only in discrete mesh-based settings. This optimization method is extremely efficient, as its computational complexity is dominated by matrix-vector multiplications using pre-computed matrices, thus may also be easily implemented on the GPU. In addition, it is quite simple to implement, and guaranteed to converge.

1.2 Previous Work

Shape deformation is one of the most active research subjects in computer graphics, and a thorough review of all the recent work is outside the scope of this paper. We shall thus concentrate on the space deformation methods most relevant to our work. In general, these methods can be classified into two major groups – "cage-based" deformation, and direct manipulation deformation.

In "cage-based" deformation, the user specifies the boundary of a relevant region of space – the source "cage" – which contains the input shape. The cage is typically a piecewise-linear closed surface. The user then manipulates the vertices of this cage to generate a target cage and the deformation is defined by the relationship between these two cages. Cage-based methods are closely related to barycentric coordinates, as typically the deformation is defined as a linear combination of the vertices of the target cage with a set of *barycentric coordinate functions* defined on the input cage. Since these barycentric coordinate functions depend only on the source cage, they can be pre-computed making for a very efficient method, as the deformation then requires only a matrix-vector multiplication. One of the first such methods [Huang et al. 2006] used mean-value coordinates [Floater et al. 2005; Ju et al. 2005] as the coordinate functions. Unfortunately, mean-value coordinates are not guaranteed to be positive inside the domain unless it is convex. This causes severe artifacts in the resulting deformation. Later methods [Joshi et al. 2007] suggested using *harmonic coordinates* instead, as these are guaranteed to be positive inside the domain. However, harmonic coordinates are the solution of a Dirichlet problem on the boundary of the domain, and they do not have a closed form expression. Thus, computing these coordinates is not easy. Recently, Lipman et al. [2008] showed how to define two sets of coordinate functions – *Green coordinates*, which have closed-form expressions, and result in detail-preserving mappings. Later, Weber et al. [2009] showed that these coordinates in two-dimensions are a special case of complex-valued barycentric coordinates, and may be derived from the celebrated Cauchy integral theorem. They called them *Cauchy-Green coordinates*. All the cage-based methods have a common disadvantage – detailed deformations are possible only with relatively complex cages, and such cages – even with a few hundred faces – are extremely hard to manipulate in order to generate a satisfying result. To overcome this problem, Weber et al. [2009] proposed something similar in spirit to our method: use the complex Cauchy-Green coordinates as *conformal basis functions*, and find the new cage location by solving an optimization problem derived from position constraints supplied by the user. Although their method is quite effective, the complex conformal formulation applies only to planar deformation. Our method can be considered as a generalization of Weber et al. [2009] to three dimensions, and R^d in general. But there is a major difference between the two methods. We use harmonic mappings as basis functions instead of conformal functions, since complex holomorphic functions (which in two dimensions generate conformal maps) do not have a simple generalization to three dimensions. As a result, in order to achieve detail-preservation, we need to solve a non-linear minimization problem, whereas in Weber et al. [2009] the optimization required the solution of a linear system.

It is worth noting that all previous methods [Joshi et al. 2007; Lipman et al. 2008; Weber et al. 2009] use harmonic maps of some sort as their underlying deformation function. Harmonic coordinates use independent harmonic functions for each coordinate, thus are able to enforce an exact interpolation of the target cage. However, this is both hard to compute, and causes serious shearing effects. Cauchy-Green coordinates in two dimensions [Lipman et al. 2008; Weber et al. 2009] enforce conformal maps – harmonic maps, whose two components have orthogonal gradients with equal norm. Green coordinates in three dimensions use the vertices of the target cage and its normals as the coefficients of a linear combination of the harmonic basis functions. From this point of view, our framework is a generalization of all those coordinates – we seek a harmonic map, but instead of predefining the relationship between its components, the relationship is derived implicitly by minimizing an energy functional.

Two other recent space-deformation methods which solve a non-linear optimization problem given positional constraints are those of Sumner et al. [2007] and Botsch et al. [2007]. In Sumner et al [2007], the deformation is defined using a *deformation graph*, which is automatically computed from the input shape. An affine transformation is associated with each node in the deformation graph, which describes the transformation this node undergoes. These transformations are the variables of an energy function – which forces them to be rigid and have a smooth behavior. Minimizing this energy, combined with the position constraints imposed by the user, generates the deformation parameters of the deformation graph. The deformation of a point in the ambient space is then computed from the transformations of nodes in the deformation graph, which are close in Euclidean distance to this point. There are two disadvantages of this method compared to ours – first, the deformation graph is not a cage, in the sense that the deformation function is computed based on Euclidean distances. This causes artifacts when deforming a shape which has pieces which are close to each other in Euclidean distance, but far apart in geodesic distance, for example, fingers of a hand. In addition, the smoothness of the deformation is enforced discretely, by requiring neighboring faces of the deformation graph to have similar transformations. In our setting, we have a closed-form expression for the second derivatives of the deformation, and we require these to vanish on the *boundary* of the domain, hence the regularization term of the energy is more robust. A similar method is that of Botsch et al. [2007], where the deformation is defined on a voxelization of the input region. Here the deformation is also computed by solving a non-linear optimization problem using a multi-grid framework. This method suffers from some aliasing effects due to the discretization, and in addition its implementation is somewhat involved. Comparisons of the results of our method with the methods of Sumner et al. and Botsch et al. will be presented in the Section 4. Other direct surface manipulation techniques exist, such as such as those of Sorkine et al. [2004], Lipman et al. [2005] and Sorkine and Alexa [2007], to mention

only a few. However, as these methods work directly on the surface of a manifold mesh, they are somewhat limited, and cannot be applied to other shape representations, such as polygon soups or point clouds.

Our formulation of the deformation mapping is based on Green's third identity, which relates the values of a harmonic function on the boundary of a region to its values *inside* the region. This is closely related both to the Green coordinates defined by Lipman et al. [2008], and to a common method for solving boundary-value problems known as the "Boundary Element Method" or BEM [Kythe 1995]. BEM has been used, for example, by Martin et al. [2008] to discretize harmonic basis functions for polyhedral finite elements. Despite the common mathematical machinery, our approach is somewhat different from both these methods. In the BEM framework, one seeks a harmonic function on the domain having some given boundary values, whereas we seek a harmonic map which minimizes a given functional. In the Green coordinates setting, the boundary mapping functions are set to be the "target cage" and its normal vectors, where as in our setting the boundary mapping functions are variables in an optimization problem.

1.3 Method Overview

Before diving into the underlying mathematics, we present a brief overview of our deformation method. The input is a polyhedral cage enclosing some region of interest, and a set of position and orientation constraints on a number of points within the cage. The output is a harmonic *deformation mapping f*, which maps every point in the input cage to some point in R^3.

As will be explained in the next section, the deformation mapping is uniquely defined by two functions, *a* and *b,* defined on the vertices and faces of the cage, respectively. In order to find *a* and *b*, we pose an optimization problem where the discrete values of *a* and *b* are the variables. The goal of the optimization problem is to minimize an energy functional which requires detail preservation and smoothness, while enforcing the user's constraints.

In the discrete setting, our method is somewhat similar to solving for the locations of the vertices of the target cage, using the Green coordinates [Lipman et al. 2008] deformation method. Thus, the function *a* is analogous to the vertex locations of the target cage, and *b* is analogous to the normals to the faces. However, there is an important difference – in our setup, the functions *a* and *b* are *independent*, whereas in the Green Coordinates setup, *b* (the normals to the faces of the target cage) are uniquely defined by *a* (the vertices of the target cage). Hence, we have more degrees of freedom, and a larger space of possible deformations.

The rest of the paper is organized as follows. In the following section we define the deformation mapping, first considering a general (continuous) domain in R^d as the cage, and then specializing it to the case where the cage is a polyhedron. We proceed by defining the energy functional, and posing the optimization problem. In Section 3 we discuss our optimization procedure, and its convergence properties. Experimental results and comparisons with state-of-the-art methods are presented in Section 4. We conclude with a discussion and some future research directions in Section 5.

2 Variational Harmonic Maps

Given the input domain – the region of space in which our shape lies, we consider all possible harmonic mappings of this domain. Within this large space of possible deformations, we will choose the harmonic map which both satisfies the user's constraints, and preserves detail as much as possible. We begin by describing our deformation mapping, first for a general domain, and then for a domain with a piecewise-linear (polyhedral) boundary. Once the deformation mapping is established, we will discuss the energy functional.

2.1 Harmonic Maps from Boundary Functions

Let Ω be an open region of R^d with a smooth boundary S, and let f be a continuous function from Ω to R^d. For example, for $d = 3$, $f = (u(x,y,z),\ v(x,y,z),\ w(x,y,z))$. We say that f is a *harmonic map* if each of its d components are harmonic functions from Ω to R. Specifically, in three dimensions, f is a harmonic map if:

$$\forall p = (x, y, z) \in \Omega, \quad \nabla^2 u(p) = 0, \nabla^2 v(p) = 0, \nabla^2 w(p) = 0$$

where ∇^2 is the Laplacian operator:

$$\nabla^2 u(x, y, z) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$$

Since the Laplacian is a linear operator, harmonic mappings form a linear subspace of functions from R^d to R^d. We would like to select a mapping from this linear space, which both satisfies the user's constraints and is detail-preserving. However, using the current formulation, it is not obvious how to find such a mapping. Fortunately, *all* harmonic maps on Ω can be generated by integrating two smooth maps defined on $S=\partial\Omega$, (the boundary of Ω) with two special functions. This is formalized in the following theorem.

Theorem: The mapping $f : \Omega \to R^d$ is a harmonic mapping if and only if there exist two C^2 mappings a and b: a,b: $S \to R^d$ such that

$$f(p) = \oint_{q \in S} a(q)(\nabla G(q, p) \cdot \hat{n}(q))dA - \oint_{q \in S} b(q)G(q, p)dA \qquad (1)$$

where $G(p,q)$ is the fundamental solution of the Laplace equation in R^d and $\hat{n}(q)$ is the unit normal direction to the surface $S=\partial\Omega$ at the point q.

Proof: Let us concentrate on the case $d = 3$. Then $G(p,q) = 1/(4\pi|p-q|)$. It is straightforward to see that f as defined in (1) is a harmonic mapping by taking the derivative relative to p under the integral sign. Let us consider the second integral – the mapping b is defined on S, hence does not depend on p. So:

$$\nabla^2 \left(b(q)G(q, p) \right) = b(q)\nabla^2 G(q, p)$$

G is a solution to the Laplace equation, hence harmonic: $\nabla^2 \left(b(q)G(q, p) \right) = 0$. Similarly, for the first integral – a is defined on the boundary and does not depend on p. So, we have:

$$\begin{aligned} \nabla^2 \left(a(q)(\nabla G(q, p) \cdot \hat{n}(q)) \right) &= a(q)\nabla^2 \left(\nabla G(q, p) \cdot \hat{n}(q) \right) \\ &= a(q)\nabla^2 \left(G_x n_x + G_y n_y + G_z n_z \right) \end{aligned}$$

where $G_x = \partial G/\partial x$, and so on, and $\hat{n}(q) = (n_x, n_y, n_z)$. Since G is harmonic, all its partial derivatives G_x, G_y and G_z are harmonic functions. In addition, the normal to the surface does not depend on p, so its dot product with ∇G is just a linear combination of harmonic functions, which is again a harmonic function.

The opposite direction is due to Green's third identity, which guarantees that any harmonic scalar function u satisfies:

$$u(p) = \oint_{q \in S} u(q)(\nabla G(q, p) \cdot \hat{n}(q))dA - \oint_{q \in S} (\nabla u(q) \cdot \hat{n}(q))G(q, p)dA \qquad (2)$$

for any point $p \in \Omega$. This is true for all the d components of f. Hence, taking

$$a(q) = f(q) \qquad b(q) = J_f(q) \cdot \hat{n}(q)$$

where $J_f(q)$ is the Jacobian of f at q, completes the proof. ♦

We will now use Eq. (1) to define our deformation mapping.

The continuous deformation mapping. The fundamental solution $G(q,p)$ to the Laplace equation has a closed-form expression for any dimension d. We define the following two scalar *kernel* functions: $\hat{\psi}, \hat{\varphi} : (S \times \Omega) \to R$

$$\hat{\psi}(q,p) = G(q,p) \qquad \hat{\varphi}(q,p) = \nabla G(q,p) \cdot \hat{n}(q)$$

Given two smooth mappings $a,b : S \to R^d$, we define the *deformation mapping* f: $\Omega \to R^d$ to be

$$f_{a,b}(p) = \oint_{q \in S} a(q)\hat{\varphi}(q,p)dA - \oint_{q \in S} b(q)\hat{\psi}(q,p)dA$$

In this way we are able to represent $f_{a,b}$ at any point of the domain as boundary integrals of the kernel functions with a and b. By the Theorem, the deformation mapping spans the linear space of all harmonic mappings on Ω, through the mappings a and b on S. Since a and b do not depend on p, we can also obtain expressions for the partial derivatives of the mapping. For example:

$$\frac{\partial f_{a,b}(p)}{\partial x} = \oint_{q \in S} a(q)\frac{\partial \hat{\varphi}(q,p)}{\partial x}dA - \oint_{q \in S} b(q)\frac{\partial \hat{\psi}(q,p)}{\partial x}dA$$

Similar expressions may be derived for any partial or higher order derivatives of f. Note that both the deformation mapping, and its derivatives, are linear in a and b. Using the deformation mapping and its derivatives, we will later define an energy functional $E(f_{a,b})$ and the final deformation of a point $p \in \Omega$ will be $f_{a',b'}(p)$ where

$$(a',b') = \arg\min(E(f_{a,b}))$$

Before defining the energy functional, we first show how the deformation mapping can be simplified in the special case that the source region is polyhedral.

The discrete deformation mapping. In the most general setup, the domain Ω, and the boundary mappings a and b, can be arbitrary. However, deformation applications usually bound Ω with a piecewise linear surface – meaning the deformed shape is contained in a d-dimensional polyhedron. In addition, we would like to restrict a and b to be of specific types, so that we can find closed expressions for the integrals of $\hat{\varphi}$ and $\hat{\psi}$ on the faces of the cage, and for their derivatives.

Note, that if a and b are restricted to a specific family of functions, one direction of the Theorem is not true anymore, and we *cannot* generate *all* harmonic mappings on Ω using (1) anymore. When choosing the families that a and b belong to, we have to make sure the identity mapping $f(p) = p$, and, more generally, any affine mapping $f(p) = Ap + T$ (where A is an $d \times d$ matrix, and T is a vector), are still obtainable by (1). In this case we should use:

$$a(q) = Aq + T \qquad b(q) = A \cdot \hat{n}(q)$$

Hence, for a piecewise-linear cage, a can be restricted to be piecewise-linear on S. Since a piecewise-linear surface has piecewise-constant normals, b can be restricted to be piecewise-constant. These are the simplest families for a and b. Of course, one could use higher degree polynomials, but then the expressions for the integrals of $\hat{\varphi}$ and $\hat{\psi}$ will be more complicated.

We will discuss now specifically the three-dimensional case for a region Ω bounded by a triangle mesh $S = (V, F)$, V are the vertices of S and F are its faces. As mentioned, a and b are not arbitrary mappings anymore – a is the piecewise-linear map on S defined by values at the vertices $\{a_v \in R^3 \mid v \in V\}$, and b is the piecewise constant map defined by values at the faces $\{b_t \in R^3 \mid t \in F\}$.
In this case, our deformation map becomes:

$$f_{a,b}(p) = \sum_{t \in F} \int_{q \in t} a(q)\hat{\varphi}(q,p)dA - \sum_{t \in F} \int_{q \in t} b_t \hat{\psi}(q,p)dA$$

Here, $a(q)$ is the piecewise linear interpolation of the values a_i, a_j, a_k on the vertices of the triangle $t = (i,j,k) \in F$.

Precisely this equation was considered by Lipman et al [2008], and the analytic solutions of the integrals were given. However, we prefer to use different expressions, which were developed in the context of boundary element methods by Urago [2000]. These expressions have a somewhat geometric interpretation, and their derivatives are easier to compute. The analytic solutions of the integrals allow us to express f using two sets of scalar functions $\{\varphi_v(p) : \Omega \to R \mid v \in V\}$, and $\{\psi_t(p) : \Omega \to R \mid t \in F\}$. Using these functions, the deformation map can be expressed as:

$$f_{a,b}(p) = \sum_{v \in V} a_v \varphi_v(p) + \sum_{t \in F} b_t \psi_t(p)$$

The expressions for φ_v and ψ_t, their gradient vectors and Hessian matrices are given in Appendix A. Figure 2 shows an example of such a deformation, given specific mappings a and b on S.

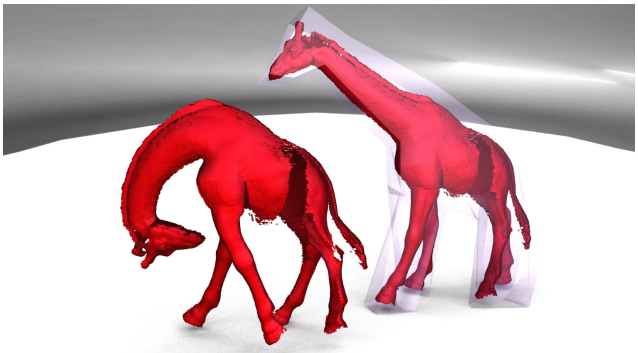


Figure 2: *Deformation of a range-scanned model (polygon soup) using our harmonic mapping. (Right) Source model enclosed in its cage. (Left) Deformed model.*

Given a point p, we can write its deformation mapping in matrix notation as follows:

$$\left(f_{\boldsymbol{a},\boldsymbol{b}}(p)\right)_{1\times 3} = \begin{pmatrix} \boldsymbol{\varphi}_{1\times n} & \boldsymbol{\psi}_{1\times m} \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_{n\times 3} \\ \boldsymbol{b}_{m\times 3} \end{pmatrix} \tag{3}$$

where n is the number of vertices, m is the number of faces, $\boldsymbol{\varphi}$ is the row vector whose entries are $\varphi_i(p)$, $\boldsymbol{\psi}$ is the row vector whose entries are $\psi_i(p)$, $\boldsymbol{a}$ is the matrix whose i-th row is a_i, and similarly for $\boldsymbol{b}$.
The transpose of the Jacobian of the deformation at the point p is:

$$\left(\boldsymbol{J}_f(p)\right)^T_{3\times 3} = \begin{pmatrix} \left(\boldsymbol{G}_\varphi\right)_{3\times n} & \left(\boldsymbol{G}_\psi\right)_{3\times m} \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_{n\times 3} \\ \boldsymbol{b}_{m\times 3} \end{pmatrix} \tag{4}$$

where $\boldsymbol{G}_\varphi$ is a matrix whose i-th column is the gradient of $\varphi_i(p)$, and similarly for $\boldsymbol{G}_\psi$.
The Hessian of the deformation at the point p is:

$$\left(\boldsymbol{H}_f(p)\right)_{5\times 3} = \begin{pmatrix} \left(\boldsymbol{H}_\varphi\right)_{5\times n} & \left(\boldsymbol{H}_\psi\right)_{5\times m} \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_{n\times 3} \\ \boldsymbol{b}_{m\times 3} \end{pmatrix} \tag{5}$$

If $p = (x,y,z)$, and $f_{a,b}(p) = (u(x,y,z), v(x,y,z), w(x,y,z))$, then the Hessian of u contains 9 values, of which only 6 are independent, due to the symmetry of the Hessian. In addition, since u is harmonic, $u_{zz} = -u_{xx} - u_{yy}$ so there are actually only 5 linearly independent values in the Hessian. These five values are present in the first column of $\boldsymbol{H}_f(p)$. The second and third columns hold the relevant

Hessian values of v and w. $\boldsymbol{H}_\varphi$ is a matrix whose i-th column holds the respective five values from the Hessian of $\varphi_i(p)$, and similarly for $\boldsymbol{H}_\psi$.

In many cases, the shape to be deformed is accompanied by normal vectors. For example, a point cloud which has a normal associated with every point, or a triangulated mesh which contains the normals of the original surface. In these cases, we would like to deform the normals as well as the shape. We can do this by deforming the plane which is orthogonal to the normal vector at the point $p \in \Omega$. Given two vectors $n_1(p)$ and $n_2(p)$, which span the plane orthogonal to $\hat{n}(p)$, we have:

$$f_{\boldsymbol{a},\boldsymbol{b}}\left(\hat{n}(p)\right) = \tilde{n}_1 \times \tilde{n}_2 = \left(\boldsymbol{J}_f(p) n_1(p)\right) \times \left(\boldsymbol{J}_f(p) n_2(p)\right)$$

where $\tilde{n}_1$ and $\tilde{n}_2$ span the deformed plane. Plugging this back into the expression for the Jacobian matrix in (4):

$$\tilde{n}_1^T(p) = n_1^T(p)\begin{pmatrix}\boldsymbol{G}_\varphi & \boldsymbol{G}_\psi\end{pmatrix}\begin{pmatrix}\boldsymbol{a}\\ \boldsymbol{b}\end{pmatrix}$$

and similarly for n_2. Hence, we can pre-compute the relevant matrices:

$$\begin{pmatrix}\tilde{n}_1^T(p)\\ \tilde{n}_2^T(p)\end{pmatrix}_{2\times 3} = \begin{pmatrix}\left(\boldsymbol{N}_\varphi\right)_{2\times n} & \left(\boldsymbol{N}_\psi\right)_{2\times m}\end{pmatrix}\begin{pmatrix}\boldsymbol{a}_{n\times 3}\\ \boldsymbol{b}_{m\times 3}\end{pmatrix} \qquad (6)$$

where $\boldsymbol{N}_\varphi$ is a matrix whose i-th column holds the dot product of $\nabla\varphi_i(p)$ with $n_1(p)$ and $n_2(p)$, and the same for $\boldsymbol{N}_\psi$.

Equipped with the deformation mapping and its first and second derivatives, we can proceed to define our energy functional, and show how to use it to find the mappings $\boldsymbol{a}$ and $\boldsymbol{b}$.

2.2 The Energy Functional

Our energy functional is similar to functionals which were used previously in "As-Rigid-As-Possible" deformation applications [Sorkine and Alexa 2007; Sumner et al 2007; Botsch et al 2007]. It attempts to satisfy the constraints specified by the user and, in addition, balance detail preservation with smoothness.

Figure 3: *Generating a realistic muscle "bulge" effect by placing a single Jacobian constraint near the marked area, and requiring it to scale. In addition to the Jacobian constraints, we have also placed position constraints causing the hand to rotate.*

User constraints. As our deformation mapping f is defined everywhere in Ω, the user can choose a set of r points $q_i \in \Omega$, and a set of s points $t_i \in \Omega$, and specify their target positions $f(q_i) = f_i$, and their Jacobians $\boldsymbol{J}_f(t_i) = \boldsymbol{g}_i$. The Jacobian constraints can be used to prescribe the orientation of the points t_i, or any other affine transform on these points. For example, in Figure 3 we have prescribed a set of position constraints, and a Jacobian constraint in the marked location, requiring its affine transform to be a small scale. This allowed us to easily generate the muscle "bulge" effect seen in the figure. The position and Jacobian constraints are hard constraints in our optimization process.

Detail vs. volume preservation. It is well known that the details of a shape at a point in space are preserved during a deformation if the local transformation that point undergoes is close to rigid. This fact has been used in many As-Rigid-As-Possible deformation methods [Botsch et al. 2007; Sumner et al. 2007; Sorkine and Alexa 2007]. However, recently Lipman et al. [2007a], have shown that detail preservation might come at the expense of volume preservation. In fact, in order to preserve the volume, Lipman et al. [2007a] scaled the transformations, according to local curvature information. Hence, requiring the Jacobians of all the points in the domain to be rotations, will not necessarily give the desired effect, and might result in volume loss.

However, in an As-Rigid-As-Possible deformation, it is reasonable to assume that the points on the *medial axis* of the domain, which is a very sparse subset of the domain itself, undergo only rotations. For example, consider Figure 4. The figure illustrates the character of the deformation of a bar to an upside-down "U" shape, similar to the deformations in Figure 8. This deformation is almost volume preserving, as its relative change in volume is 0.04. The figure color-codes the determinant and condition number of the Jacobian of the deformation on a vertical slice through the shape. The determinant indicates the local change in volume, and the condition number ($\sigma_{max}/\sigma_{min}$) indicates the amount of non-uniform scale. As evident in the figure, the volume on the top of the bar increases, the volume on the bottom the bar decreases, but the medial axis of the bar is only bent – the volume near it remains constant. In addition, the condition number is closest to 1 near the medial axis, which indicates that the transformations in this area are close to rotations.

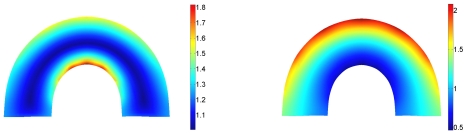


Figure 4: *The character of the Jacobian of the deformation within one slice through a vertical bar model, bent to a "U" shape. (left) Color-coding of the condition number. (right) Color-coding of the determinant.*

In our setting, the local transformation of a point p is simply the Jacobian matrix of $f_{\boldsymbol{a},\boldsymbol{b}}$ at p. Since we can prescribe the Jacobians of the deformation in any location we choose, we prescribe the Jacobians of the medial axis to be as close as possible to rotations. This way we don't need to compute the desired transformations on the boundary of the domain, as they will be implied from the smoothness of the deformation. The values of these rotations are not known in advance, and will be computed as part of the optimization process.

Hence, we would like to minimize the following rigidity energy:

$$\min_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{R}(\Omega)} E_{Rigid}(f_{\boldsymbol{a},\boldsymbol{b}}) = \int_{p\in M(\Omega)} \left\|\boldsymbol{J}_f(p) - \boldsymbol{R}(p)\right\|_F^2 d\omega$$

$$s.t.\ \ \forall p \in M(\Omega)\quad \boldsymbol{R}(p)^T \boldsymbol{R}(p) = \boldsymbol{I}$$

The Rigidity Energy

where $M(\Omega)$ is the medial axis of the domain and the norm is the Frobenius matrix norm. The Jacobian is linear in the variables $\boldsymbol{a}$ and $\boldsymbol{b}$, hence if we knew which rotations $\boldsymbol{R}(p)$ each point should undergo, minimizing E_{Rigid} would be a simple matter of minimizing a quadratic energy. Of course, $\boldsymbol{R}(p)$ are not known in advance, hence the optimization process is non-linear.

Smoothness. The local transformation of a point $p \in \Omega$ is governed by the Jacobian of the mapping f at p. A smooth deformation will have similar transformations for nearby points, and hence a small second derivative. So, to enforce smoothness, we require the Frobenius norm of the Hessian matrix of each of the

deformation mapping components to be as small as possible, by minimizing the following energy:

$$\min_{\boldsymbol{a},\boldsymbol{b}} E_{Smooth}(f_{\boldsymbol{a},\boldsymbol{b}}) = \int_{p\in\Omega} \left\| \boldsymbol{H}_f(p) \right\|_F^2 d\omega$$

This energy can be simplified using the following observation. Our mapping is harmonic, and hence all the partial and higher derivatives of all its components are also harmonic. According to the maximum principle, a harmonic function on a domain achieves its extremum on the boundary of the domain. Hence, if we minimize the second derivatives on the *boundary* of the domain, they will also be bounded *inside* the domain. As a result, we may use the following smoothness energy:

$$\min_{\boldsymbol{a},\boldsymbol{b}} E_{Smooth}(f_{\boldsymbol{a},\boldsymbol{b}}) = \int_{p\in S} \left\| \boldsymbol{H}_f(p) \right\|_F^2 ds$$

The Smoothness Energy

The energy. Given the points q_i and t_i chosen by the user, and their target positions f_i and target Jacobians $\boldsymbol{g}_i$ respectively, we would like to solve the following optimization problem:

$$\min_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{R}(p)} E(f_{\boldsymbol{a},\boldsymbol{b}}) = \int_{p\in M(\Omega)} \left\| \boldsymbol{J}_f(p) - \boldsymbol{R}(p) \right\|_F^2 d\omega + \lambda^2 \int_{p\in S} \left\| \boldsymbol{H}_f(p) \right\|_F^2 ds$$

$$s.t. \quad \forall i = 1..r, \quad f_{a,b}(q_i) = f_i \quad , \quad \forall i = 1..s, \quad \boldsymbol{J}_f(t_i) = \boldsymbol{g}_i$$

$$\forall p \in M(\Omega), \qquad \boldsymbol{R}(p)^T \boldsymbol{R}(p) = I$$

The Continuous Optimization Problem

where the norms are Frobenius norms, and $\boldsymbol{R}(p)$ are unknown 3×3 matrices defined on every point p on the medial axis of Ω.

The discrete energy. Minimizing the energy functional in its current form is difficult, because of the non-linearity of the rotation constraints, and because we do not have closed-form expressions neither for the medial axis, nor for the integrals. Instead, we convert the integrals to a sum of finite samples, as follows. For the smoothness energy, we sample the boundary surface S at k points w_i. For the rigidity energy we approximate the medial axis, by sample points on a set of *rigidity lines*. These lines can be acquired from a skeleton of the deformed shape if it is available, can be prescribed manually by the user, or can be computed using a skeleton extraction algorithm, such as that of Au et al. [2008]. Once l such lines are given, we sample them at d *anchor* points m_i, by sampling d/l points on each rigidity line.

Since we require smoothness, it is sufficient for the anchor points to be sparsely distributed, so d can be relatively small. The assumption that a sparse set of rigidity constraints is enough when the deformation is smooth has been used successfully in other deformation methods [Weber et al. 2007; Sorkine and Cohen-Or 2004]. Consequently, we only have d unknown rotation matrices $\boldsymbol{R}_i$ to solve for. In this setting the optimization problem becomes:

$$\min_{\boldsymbol{a},\boldsymbol{b},\boldsymbol{R}_i} E(f_{\boldsymbol{a},\boldsymbol{b}}) = \sum_{i=1}^{d} \left\| \boldsymbol{J}_f(m_i) - \boldsymbol{R}_i \right\|_F^2 + \lambda^2 \sum_{i=1}^{k} \left\| \boldsymbol{H}_f(w_i) \right\|_F^2$$

$$s.t. \quad \forall i = 1..r, \quad f_{a,b}(q_i) = f_i \quad , \quad \forall i = 1..s, \quad \boldsymbol{J}_f(t_i) = \boldsymbol{g}_i$$

$$\forall i = 1..d, \quad \boldsymbol{R}_i^T \boldsymbol{R}_i = \boldsymbol{I}$$

(P1): The Discrete Optimization Problem

Figure 5 shows a comparison of results using a different number of anchor points for the rigidity constraints. The locations of the anchor points were computed using the skeleton extraction algorithm by Au et al. [2008]. As is evident from the figure, increasing the number of anchors beyond a given point does not significantly improve the results.

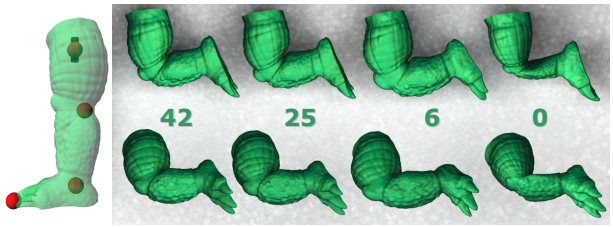


Figure 5: *Deformation using a different number of anchor points. The leg of the armadillo model (left) was deformed to a bent position, using the specified number of anchor points. The top and bottom rows show different views of the same deformed shape. The source pose shows the five user constraints – red spheres are positional constraints, and black cylinders are orientation constraints.*

Figure 6 shows two deformations of the "Armadillo" model. In addition, the figure shows the setup for the deformation – the cage, the original pose, the anchor points and the constraints.

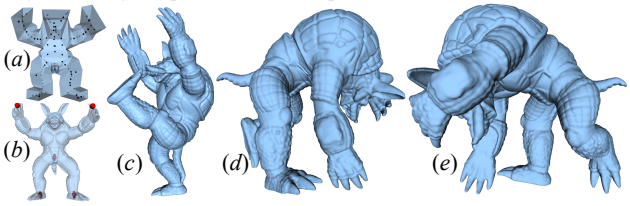


Figure 6: *Deformations of the Armadillo model (a) Cage and anchor locations (b) Original pose and constraints (c) Deformed pose (d,e) Another deformed pose from two different viewpoints*

In the following section we describe our optimization scheme for minimizing the deformation energy.

3 Optimization

To solve the optimization problem (P1) we use the following observation. If the variables $\boldsymbol{R}_i$ are known, then (P1) is a simple linear least-squares problem with linear equality constraints, which has a closed-form global minimum. On the other hand, if $\boldsymbol{a}$ and $\boldsymbol{b}$ are known, then the optimal rotation matrices $\boldsymbol{R}_i$ – those which are closest in Frobenius norm to the Jacobians of the deformation map at m_i - also have a closed-form solution. This solution is a variant of the well-known "Procrustes problem", obtained using Singular Value Decomposition (SVD). Hence, we can solve (P1) using the *alternating least squares* method, or "local/global" algorithm [Liu et al 2008; Sorkine and Alexa 2007]. In the "local" step, we keep $\boldsymbol{a}$ and $\boldsymbol{b}$ fixed, and solve many small and independent local problems for the $\boldsymbol{R}_i$, while in the "global" step, we keep $\boldsymbol{R}_i$ fixed and solve one global linear system for $\boldsymbol{a}$ and $\boldsymbol{b}$. We repeat these two steps until convergence.

Convergence and robustness. As was pointed out in previous works [Liu et al 2008; Sorkine and Alexa 2007], the "local/global" algorithm is guaranteed to converge, because each step must reduce the energy. In general, the convergence rate depends on the initial configuration. However, since the number of variables is relatively small – the number of anchors for the Jacobian computation is usually smaller than the complexity of the cage – the "local/global" algorithm is robust enough to converge to a good solution from an arbitrary initial configuration. By "arbitrary" - we mean that the Jacobians are initialized to be random 3×3 matrices.

Figure 7 shows the resulting deformation after various numbers of iterations, starting from an arbitrary configuration. In addition, it

shows graphs of the value of the energy functional vs. the iteration number using different initial configurations. As can be seen from the graph, our method always converged to the same solution, no matter which initial configuration was used.

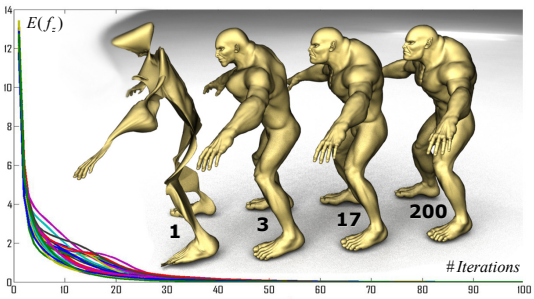


Figure 7: *The "local/global" optimization scheme is robust enough to converge to a good solution from any arbitrary initial configuration. (Left to right) the deformed shape after 1, 3, 17 and 200 iterations, starting from an arbitrary initial configuration. The graphs show the value of the energy functional vs. the number of iterations, starting from different random starting points.*

In an interactive modeling environment, the initial configuration can be taken from the values of $\boldsymbol{a}$,$\boldsymbol{b}$ and $\boldsymbol{R}_i$ in the previous frame. In this case, a small number of iterations of the "local/global" algorithm are usually enough to achieve convergence. This is the initial configuration we used for all the examples in the paper, except the ones in Figure 7. In Section 4 we provide some more deformation examples, with the timings required to generate them.

Let us now turn to a more detailed description of the minimization process.

Implementation details. Using the "local/global" approach, the deformation algorithm is relatively simple to implement, and boils down to three steps – the preprocessing step, during which some matrices are pre-computed for later use, the optimization step, where we iterate the "local/global" steps to find the values of $\boldsymbol{R}_i$, and, finally, the deformation step, during which the values of $\boldsymbol{R}_i$ are combined with the user's constraints to generate the final mapping of the input shape. These steps are implemented as a series of matrix operations, on matrices which are "stacked" matrices of $\boldsymbol{\varphi}$, $\boldsymbol{\psi}$ and their derivatives. To avoid clutter in the notation, we redefine (3), (4) and (5) in terms of single matrices as follows:

$$f_{\boldsymbol{a},\boldsymbol{b}}(p) = \boldsymbol{D}_p \boldsymbol{z} \quad , \quad \boldsymbol{J}_f(p) = \boldsymbol{J}_p \boldsymbol{z} \quad , \quad \boldsymbol{H}_f(p) = \boldsymbol{H}_p \boldsymbol{z}$$

where each matrix represents a concatenation of matrices from (3), (4) and (5) respectively: $\boldsymbol{D}_p = [\boldsymbol{\varphi}, \boldsymbol{\psi}]$, $\boldsymbol{J}_p = [\boldsymbol{G}_\varphi, \boldsymbol{G}_\psi]$, $\boldsymbol{H}_p = [\boldsymbol{H}_\varphi, \boldsymbol{H}_\psi]$. In addition, $\boldsymbol{a}$ is a matrix of size $n\times3$, and $\boldsymbol{b}$ is a matrix of size $m\times3$ (where n and m are the number of vertices and faces respectively). $\boldsymbol{z}$ is the matrix whose first n rows are $\boldsymbol{a}$, and last m rows are $\boldsymbol{b}$.

Now we can convert the optimization problem to matrix notation using these expressions:

$$\min_{\boldsymbol{z},\hat{\boldsymbol{R}}} E(f_z) = \left\| \hat{\boldsymbol{J}}\boldsymbol{z} - \hat{\boldsymbol{R}} \right\|_F^2 + \lambda^2 \left\| \hat{\boldsymbol{H}}\boldsymbol{z} \right\|_F^2$$

$$s.t. \quad \hat{\boldsymbol{D}}\boldsymbol{z} = \hat{\boldsymbol{f}}, \quad \tilde{\boldsymbol{J}}\boldsymbol{z} = \hat{\boldsymbol{g}} \quad \forall i = 1..d, \quad \boldsymbol{R}_i^T \boldsymbol{R}_i = \boldsymbol{I}$$

where:

$$\hat{\boldsymbol{D}}_{r\times(n+m)} = \begin{pmatrix} \boldsymbol{D}_{q_1} \\ \cdots \\ \boldsymbol{D}_{q_r} \end{pmatrix} \quad \tilde{\boldsymbol{J}}_{3s\times(n+m)} = \begin{pmatrix} \boldsymbol{J}_{t_1} \\ \cdots \\ \boldsymbol{J}_{t_s} \end{pmatrix} \quad \hat{\boldsymbol{J}}_{3d\times(n+m)} = \begin{pmatrix} \boldsymbol{J}_{m_1} \\ \cdots \\ \boldsymbol{J}_{m_d} \end{pmatrix} \quad \hat{\boldsymbol{H}}_{5k\times(n+m)} = \begin{pmatrix} \boldsymbol{H}_{w_1} \\ \cdots \\ \boldsymbol{H}_{w_k} \end{pmatrix}$$

are stacks of deformation, Jacobian and Hessian matrices for the respective points (r position constraints, s orientation constraints, d anchor points and k Hessian sample points on the boundary of the domain), and:

$$\hat{\boldsymbol{f}}_{r\times3} = \begin{pmatrix} f_1 \\ \cdots \\ f_r \end{pmatrix} \quad \hat{\boldsymbol{g}}_{3s\times3} = \begin{pmatrix} \boldsymbol{g}_1 \\ \cdots \\ \boldsymbol{g}_s \end{pmatrix} \quad \hat{\boldsymbol{R}}_{3d\times3} = \begin{pmatrix} \boldsymbol{R}_1 \\ \cdots \\ \boldsymbol{R}_d \end{pmatrix}$$

are the right hand sides of the linear equations – the user's position and orientation constraints, and the unknown rotation matrices $\boldsymbol{R}_i$ at the anchor points. The energy can now be written as:

$$E = \left\| \boldsymbol{A}\boldsymbol{z} - \begin{pmatrix} \hat{\boldsymbol{R}} \\ 0 \end{pmatrix} \right\|_F^2 \quad , \quad \boldsymbol{A}_{(3d+5k)\times(n+m)} = \begin{pmatrix} \hat{\boldsymbol{J}} \\ \lambda\hat{\boldsymbol{H}} \end{pmatrix}$$

The constant λ determines the relative weight of the smoothness constraints vs. the rigidity constraints. In our experiments, we took λ to be: $\lambda = \alpha d / |\hat{\boldsymbol{J}}|_\infty / |\hat{\boldsymbol{H}}|_\infty$. The matrix norms are infinity norms – the maximal L_1 norms of the rows of the matrix, and α is a user specified parameter, which can be used to control the stiffness of the deformation. We took α to be 0.01 in all of our experiments.

Returning to the optimization problem, if $\hat{\boldsymbol{R}}$ is known, then the minimum of E is given by:

$$\boldsymbol{z}_{opt} = \boldsymbol{A}^+ \begin{pmatrix} \hat{\boldsymbol{R}}_{3d\times3} \\ 0_{5k\times3} \end{pmatrix} \quad , \quad \boldsymbol{A}^+ = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T$$

Since most of the columns of $\boldsymbol{A}^+$ are multiplied by zero on the right hand side, we can truncate its last $5k$ columns as follows:

$$\boldsymbol{z}_{opt} = \boldsymbol{A}^+_{trunc}\hat{\boldsymbol{R}} \quad , \quad \boldsymbol{A}^+_{trunc} = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T_{trunc}$$

where $\boldsymbol{A}^T{}_{trunc}$ includes only the first $3d$ columns of $\boldsymbol{A}^T$.

We enforce the user's constraints by removing r+3s variables from the problem, and computing their value from the remaining variables using the equations:

$$\begin{pmatrix} \hat{\boldsymbol{D}} \\ \tilde{\boldsymbol{J}} \end{pmatrix} \boldsymbol{z}_{opt} = \hat{\boldsymbol{h}} \quad , \quad \hat{\boldsymbol{h}}_{(r+3s)\times3} = \begin{pmatrix} \hat{\boldsymbol{f}} \\ \hat{\boldsymbol{g}} \end{pmatrix}$$

This can be done, of course, only if the number of hard constraints is less than the number of degrees of freedom in the problem – n+m. However, relatively complicated deformations can be generated with a small number of position and orientation constraints. The resulting system is of the type:

$$\boldsymbol{z}_{opt} = \boldsymbol{B}\begin{pmatrix} \hat{\boldsymbol{h}} \\ \hat{\boldsymbol{R}} \end{pmatrix}$$

where $\boldsymbol{B}$ is computed from partial matrices of $\boldsymbol{A}$. During the optimization procedure, we need to re-compute the current Jacobian matrices. Hence, we get:

$$\hat{\boldsymbol{R}}_{new} = \boldsymbol{C}\begin{pmatrix} \hat{\boldsymbol{h}} \\ \hat{\boldsymbol{R}} \end{pmatrix}, \quad \boldsymbol{C} = \hat{\boldsymbol{J}}\boldsymbol{B}$$

In addition, once the non-linear iteration has converged, we need to compute the new location of the deformed shape. The new location of the points $x_1,x_2,...,x_a$ are given by:

$$\begin{pmatrix} \tilde{x}_1 \\ \cdots \\ \tilde{x}_a \end{pmatrix} = \begin{pmatrix} \boldsymbol{D}_{x_1} \\ \cdots \\ \boldsymbol{D}_{x_a} \end{pmatrix} \boldsymbol{z}_{opt} = \begin{pmatrix} \boldsymbol{D}_{x_1} \\ \cdots \\ \boldsymbol{D}_{x_a} \end{pmatrix} \boldsymbol{B}\begin{pmatrix} \hat{\boldsymbol{h}} \\ \hat{\boldsymbol{R}}_{opt} \end{pmatrix} = \boldsymbol{F}\begin{pmatrix} \hat{\boldsymbol{h}} \\ \hat{\boldsymbol{R}}_{opt} \end{pmatrix} \quad , \quad \boldsymbol{F} = \begin{pmatrix} \boldsymbol{D}_{x_1} \\ \cdots \\ \boldsymbol{D}_{x_a} \end{pmatrix} \boldsymbol{B}$$

The matrices $\boldsymbol{C}$ and $\boldsymbol{F}$ are pre-computed before the interactive deformation begins. Thus, we have laid out all the building blocks for our algorithm, which can be stated as follows:

Pre-processing. Compute the matrices $\boldsymbol{C}$ and $\boldsymbol{F}$, given the locations of the user's constraints, the anchor points, the Hessian samples on the boundary and the input shape.

Optimization. Select an initial solution $\boldsymbol{z}$, $\hat{\boldsymbol{R}}$, and set $\hat{\boldsymbol{R}}_{Global} = \hat{\boldsymbol{R}}$. Repeat until convergence the following two steps:

1. $\hat{\boldsymbol{R}}_{Local} = normalize(\hat{\boldsymbol{R}}_{Global})$
2. $\hat{\boldsymbol{R}}_{Global} = \boldsymbol{C}\begin{pmatrix}\hat{\boldsymbol{h}} \\ \hat{\boldsymbol{R}}_{Local}\end{pmatrix}$ (7)

In the local step, the "normalization" of the matrices $\hat{\boldsymbol{R}}_{Global}$ is done by computing the SVD for each matrix $\boldsymbol{R}_i = USV^T$, and replacing it with UV^T, up to a change of sign in the last column of U, if it has a negative determinant. An additional benefit of this local step, is that since Jacobians with negative determinant are not allowed, the optimization process tends to find a minimum which doesn't contain foldovers. Of course, since this might be overridden by the global step, the occurrence of foldovers depends on the user's constraints. In our experience, for a reasonable set of constraints, foldovers are not likely to appear.

We detect convergence by measuring the amount of change in $\hat{\boldsymbol{R}}_{Global}$ between two consecutive iterations, which is equivalent to the change in the rigidity energy. It would be better to measure the change in the total energy, however this is more computationally expensive. The computational complexity of each iteration is the complexity of computing d SVD operations, and a matrix-vector multiplication which is $O(d(r+3s+d))$. Detailed performance timings are provided in the next section. Note that the computational complexity of both the local and the global shape *do not* depend on the complexity of the deformed shape, nor on the complexity of the cage.

Deformation. If $\hat{\boldsymbol{R}}_{opt}$ is the last $\hat{\boldsymbol{R}}_{Global}$ computed in the optimization step, then the deformed locations are given by:

$$\begin{pmatrix}\tilde{x}_1 \\ \dots \\ \tilde{x}_a\end{pmatrix} = \boldsymbol{F}\begin{pmatrix}\hat{\boldsymbol{h}} \\ \hat{\boldsymbol{R}}_{opt}\end{pmatrix} \quad (8)$$

The algorithm can be summed up in a few lines of pseudo-code, outlined in Algorithm 1. The pre-process step requires only vector and matrix operations to set up the matrices, and multiply them. Hence, using any efficient linear algebra package, the implementation is relatively straightforward. We provide runtimes of all the steps of the algorithm, for various 3D models, in the next section.

```
Precompute C,F
While (err > threshold) do
       Js_prev = Js
       Js = normalize_jacobians(Js)
       Js = C*[constraints;Js]
       err = norm(Js_prev - Js)
end
new_positions = F*[constraints;Js]
```

Algorithm 1: *Pseudo-code of the deformation algorithm*

4 Experimental Results

We implemented our "Variational Harmonic Map" (VHM) deformation system as a plugin to the Maya® commercial modeling and animation system. The optimization and deformation step of VHM include two building blocks – SVD computations of 3×3 matrices, and *dense* matrix-vector multiply. Dense matrix-vector multiply operations are "*embarrassingly parallel*" in the sense that they are composed of many independent operations (multiplying one row by one column), which can be performed in parallel. We have exploited this by implementing the computation of Equations (7) and (8) on the GPU. We used Nvidia's CUDA programming language with the BLAS library, on an Nvidia Quadro FX 5800 graphics card. Figures 1-3, 6-14 and the accompanying video demonstrate the application of VHM to different deformation scenarios. In this section we will first compare VHM to two other state-of-the-art deformation methods, and then discuss some of its properties.

Comparison. We compared the performance of VHM to two state-of-the-art deformation methods: "Embedded Deformation" (ED) of Sumner et al. [2007] and "Adaptive Rigid Cells" (ARC) of Botsch et al. [2007]. We compared these methods on three deformation scenarios of a synthetic model, and on one deformation of the beast model – measured by the overall appearance of the deformed shape, the detail preservation and the change in the total volume of the shape. Software was kindly provided by the respective authors.

Before starting the comparison we should state upfront some disadvantages of VHM. Its biggest downside, compared to ED and ARC, is that in additional to the shape to be deformed, the user must also supply a cage bounding the domain, and a set of "rigidity lines". Although generating the rigidity lines is relatively painless (e.g. using a skeleton extraction algorithm such as [Au et al. 2008]), creating a cage is not a trivial problem, and this is mostly understated in existing cage-based deformation methods. In this respect, methods which automatically generate the underlying space representation – the deformation graph for ED and the voxelization for ARC, have an advantage. On the other hand, we believe the benefits of having a cage – a closed form expression for the deformation, faster optimization and separation of unrelated parts of the shape – outweigh the hassle of generating such a cage.

Figure 8 shows a comparison between VHM, ED and ARC for the "bar" shape, with three different deformations. For VHM and ED methods, we used the same constraints. For ARC, we achieved the deformation through interactive manipulation. The results shown for ARC are after the final RBF interpolation step.

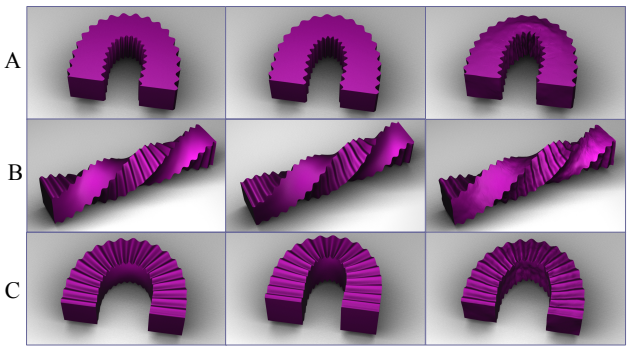


Figure 8: *Comparison of our deformation method – VHM - with ARC and ED on three deformations of the "bar" model.*

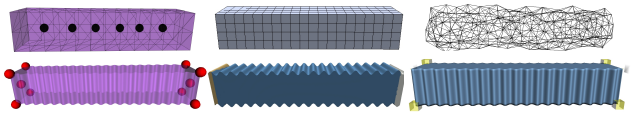


Figure 9: *The setup used for the comparisons in Figure 8. (top, from left to right) VHM cage and anchors, ARC cells (320) and ED deformation graph (187 vertices). (bottom) The VHM, ARC and ED constraints.*

Figure 9 shows the setup we used for the deformations in Figure 8. Figure 10 shows the comparison and setup for the deformation

of the beast model. The models were interactively deformed to reach the required pose.

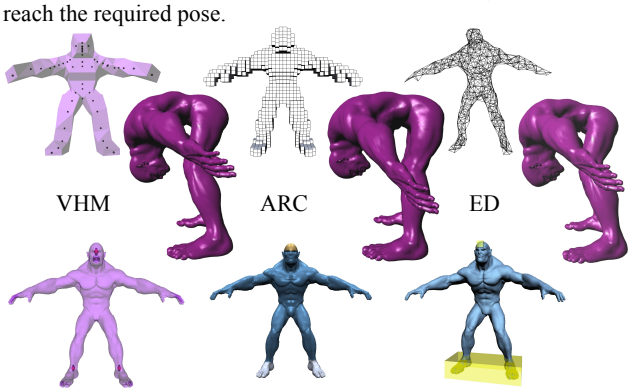


Figure 10: *Comparison of our method (VHM) with ARC and ED on a deformation of the "Beast" model, and the setup used for the deformation. Small images: (top) VHM cage and anchors, ARC cells (2148) and ED deformation graph (300 vertices), (bottom) the VHM, ARC and ED constraints.*

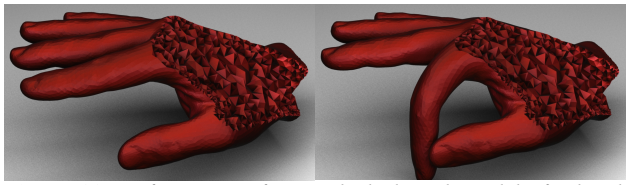


Figure 11: *Deformation of a tetrahedral mesh model of a hand. One finger is easily moved without influencing the nearby finger, even though it is close in Euclidean distance.*

To compare the detail preservation of the different models, we computed the *rigidity distortion* of the triangles of the deformed mesh, which is defined similarly to Liu et al. [2008] as:

$$E_{Rigid} = \sum_{t=1}^{m} A_t \left[\left(\sigma_{1,t} - 1\right)^2 + \left(\sigma_{2,t} - 1\right)^2 \right] \Big/ \sum_{t=1}^{m} A_t$$

where A_t is the area of the source triangle, and $\sigma_{1,t}$ and $\sigma_{2,t}$ are the singular values of the Jacobian of the 2×2 transformation, that transforms the source planar triangle to the deformed planar triangle. Ideally, we would like to compare the singular values of the Jacobian of the 3D transformation, but since for the other two methods we do not have access to the actual deformation function, rather only the end result, this is, unfortunately, not easily done. In addition, we compared the change in the total volume of the deformed shape as:

$$E_{volume} = \left| vol_{new} - vol_{orig} \right| \Big/ vol_{orig}$$

The comparison of these errors is given in Table 1.

	E_{rigid}			E_{volume}		
Model	VHM	ARC	ED	VHM	ARC	ED
A	0.050	0.035	0.049	0.086	0.078	0.143
B	0.069	0.070	0.078	0.177	0.116	0.226
C	0.046	0.043	0.053	0.069	0.082	0.118
Beast	0.022	0.013	0.018	0.063	0.025	0.119

Table 1: *Comparison of the rigidity error and volume change of the deformation methods.*

As can be seen from Figures 8 and 10, and Table 1, the results of VHM are comparable to those of ARC, however VHM is considerably more efficient (as is shown in Table 2), and also simpler to implement. When compared to ED, our method is somewhat better, both in the visual quality of the results – Figure 8 shows that the ED method has some noise issues - and in volume preservation.

Locality of the deformation. Figure 11 demonstrates that VHM has a local effect, and only regions geodesically close to the manipulated regions are modified, as opposed to unrelated regions which happen to be close in Euclidean distance – the index finger of the hand may be moved without influencing the other fingers.

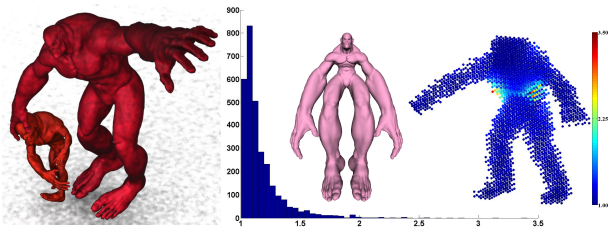


Figure 12: *(left) Two As-Similar-As-Possible deformations of the Beast model. Note the exaggerated hands and feet.(right) Another ASAP deformation, and the color coding of the condition number of the Jacobian of the deformation, sampled on the input cage. The graph shows the histogram of these values.*

As-Similar-As-Possible deformations. One of the benefits of our "local/global" optimization scheme is that the constraints on the Jacobian matrices of the anchor points can be easily changed from rigidity constraints to other types of constraints, simply by modifying the local step in the optimization algorithm. For example, as was done in previous "local/global" based methods, such as [Liu et al. 2008], we can replace the rigidity constraints with similarity constraints by requiring the Jacobian matrices of the anchor points to be similarity transforms. The local step is modified by replacing the "normalization" step of the Jacobian matrices with the following procedure: Compute the SVD for each matrix $\boldsymbol{R}_i = USV^T$, and replace it with $US_{new}V^T$, where S_{new} is a diagonal matrix, whose entries are the average of the diagonal entries of S. Such a deformation will not be As-Rigid-As-Possible anymore, as it introduces uniform scale. However, as can be seen in Figure 12 and in the accompanying video, interesting exaggeration effects can be generated this way.

For some applications, one might require the deformation to be quasi-conformal, meaning that the condition number of the Jacobian of the deformation is bounded. In these cases, the As-Similar-As-Possible approach is more appropriate than the As-Rigid-As-Possible approach. Figure 12 shows the color-coding of the condition number of the Jacobian, for sampled points inside the cage of the Beast model, for the shown deformation. In addition, the figure shows the histogram of these values. As is evident from the figure, the condition numbers are smaller than 3.5, which indicates that this deformation is quasi-conformal, with a quasi-conformal factor similar to the that of the Green coordinates [Lipman et al. 2008].

The cage. As opposed to direct manipulation methods [Botsch et al. 2007, Sumner et al. 2007], which build the underlying representation automatically, cage based methods such as [Lipman et al. 2008] usually rely on a manually modeled cage. We also use manually modeled cages, but since in our approach the cages are only a mathematical tool, and are not visible to the user, it is important to check how sensitive the deformation is to the cage used. Specifically, we would like to verify that two reasonable cages result in similar deformations, when the user constraints are identical. To investigate this, we implemented a straightforward algorithm to generate a simple cage by uniform decomposition of

space, followed by merging neighboring co-planar faces. Such cages would be somewhat hard to manipulate manually, but since in our method the user does not manipulate the cage directly, this is not an issue. We applied this algorithm to the "Beast" model from Figure 1, and using the manually built and automatic cages, we deformed the model interactively. Figure 13 shows the two cages, and the deformations resulting from them. As is evident from the figure, the deformations induced by the two cages are very similar, indicating that our method is not very sensitive to the precise cage used.

Figure 13: *Two deformations using a manually built cage (left), and an automatic cage (right)*

Non-articulated shapes. Some objects, such as plate-like objects, do not have an obvious skeleton. In these cases, our method can still be applied by placing the anchors on the *medial surface* instead of on the medial axis. Figure 14 shows two deformation of a "Bumpy plane" model, using different anchors configurations. In both cases, the anchors were placed on the medial surface of the model, but their exact placement was different. As the figure shows, the resulting deformations are very similar, indicating that our method is not very sensitive to the exact locations of the anchors on the medial surface. The figure also shows the comparison of the results to the deformation of the same model using the ARC method.

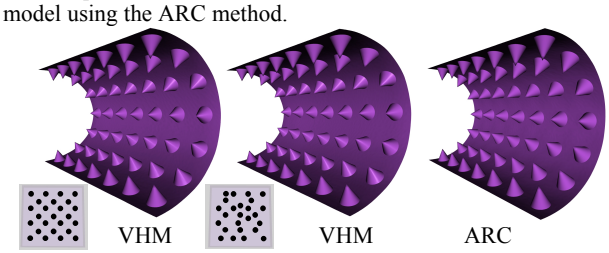


Figure 14: *Deformation of a plate like object, using two different anchor configurations on the medial surface (left and middle). Deformation using ARC of the same model (right)*

Efficiency. Table 2 provides the model statistics and the deformation times in milliseconds for our examples. The deformation timing is broken down into the time for *one* optimization iteration (labeled "Solve") and the time for the matrix-vector multiply which generates the deformation (labeled "Def"). The preprocessing time for all the models was less than a minute. It is clear from the table that our solve times are considerably faster than those reported for ARC [Botsch et al. 2007] and ED [Sumner et al. 2007], which were run on machines with spec similar to ours. For example, the solve step of ED for the Giraffe model requires 120 msecs, using six Gauss-Newton iterations. Using the ARC method, the solve step for a model with 50,000 vertices requires 330 msecs for a *single* Newton iteration. For a larger model of 79,000 vertices, the solve step of VHM requires only 12 msecs. The VHM solve includes GPU optimization (for the global part of the "local/global" algorithm), whereas the other methods are implemented on the CPU. However, the ARC and ED optimization algorithms are based on Gauss-Newton iterations using a large *sparse* matrix. Such algorithms are considerably harder to parallelize than dense matrix-vector multiplies, which can be implemented using off-the-shelf CUDA code. Hence, if one is to compare the best possible implementation of the methods, ours has a distinct advantage. The deformation times are also very fast, with 10 msecs for the 170,000 vertex Armadillo model.

Model	Verts	Cage faces	Ancrs	Iters	Solve (ms)	Def (ms)	Tot (ms)
Bar	32,908	208	6	15	0.27	1.84	5.89
Tet Hand	28,796	288	28	9	0.43	2.27	6.14
Giraffe	79,226	204	27	33	0.37	3.08	15.29
Beast	32,311	226	50	10	0.53	2.60	7.90
Arma leg	28,829	68	6	26	0.27	1.87	8.89
Arma	173,101	250	88	13	0.69	10.60	19.57

Table 2: *Performance measured in msecs on an Intel 2.67GHz i7 machine (using a single thread) with 4GB of RAM. "Solve" - time for one optimization iteration, "Def" - time for the matrix multiply in the deformation step. "Iters" - average number of iterations a typical deformation requires to converge.*

5 Conclusions and Discussion

We have proposed a new space deformation method ("Variational Harmonic Mapping" – VHM) whose underlying mathematical model is a harmonic mapping. Using this mapping and its derivatives, we defined an energy function whose minimization allows the user to deform the shape using a small number of position and orientation constraints. We showed how to minimize the energy using a very efficient iterative "local/global" algorithm and demonstrated that the resulting deformation is close to an As-Rigid-As-Possible deformation. Its quality is comparable to state-of-the-art space deformation methods, while being considerably faster. In the future we hope to further explore variational harmonic mappings in settings other than deformation. Due to the similarity to boundary element methods (BEM) [Kythe 1995], our method might also be effective in finding solutions to different interpolation problems. Moreover, we would like to investigate the theoretical properties of our deformation, and its relation to quaternionic analytic functions.

Acknowlegments

We would like to thank Robert Sumner, Mario Botsch, Oscar Kin-Chung Au, Daniel Cohen-Or and Amit Mano for supplying us with their software implementations, Robert Sumner and Mark Pauly for the giraffe model, Autodesk for the Beast model, the AIM@SHAPE project for the Armadillo and hand models, and NVIDIA for the donation of the Quadro graphics card. This work was partially supported by Israel-Niedersachsen (Volkswagen Foundation) grant #ZN2046, the Israel Ministry of Science and the Fund for the Promotion of Research at the Technion.

References

AU, O. K.-C., TAI, C.-L., CHU, H.-K., COHEN-OR, D., AND LEE, T.-Y. 2008. Skeleton extraction by mesh contraction. *ACM Trans. Graph.* 27, 3, 1-10.

BOTSCH, M., PAULY, M., WICKE, M., AND GROSS, M. 2007. Adaptive space deformations based on rigid cells. *Computer Graphics Forum* 26, 3, 339-347.

DONG, S., KIRCHER, S., AND GARLAND, M. 2005. Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometric Design* 22, 5, 392-423.

FLOATER, M.S., KÓS, G., AND REIMERS, M. 2005. Mean value coordinates in 3D. *Computer Aided Geometric Design* 22, 7, 623–631.

FLOATER, M. S. AND HORMANN, K. 2005. Surface parameterization: A tutorial and survey. *Advances in Multiresolution for Geometric Modeling* 157-186.

HUANG, J., SHI, X., LIU, X., ZHOU, K., WEI, L.-Y., TENG, S.-H., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Subspace gradient domain mesh deformation. *ACM Trans. Graph.* 25, 3, 1126-1134.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph.* 26, 3, 71.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.* 24, 3, 561-566.

KYTHE, K., P. 1995. *An Introduction to Boundary Element Methods*. CRC Press.

LIPMAN, Y., COHEN-OR, D., GAL, R., AND LEVIN, D. 2007. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.* 26, 1, 5.

LIPMAN, Y., KOPF, J., COHEN-OR, D., AND LEVIN, D. 2007. GPU assisted positive mean value coordinates for mesh deformations. In *Proc. Symposium on Geometry Processing*, 117-123.

LIPMAN, Y., LEVIN, D., AND COHEN-OR, D. 2008. Green coordinates. *ACM Trans. Graph.* 27, 3, 1-10.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.* 24, 3, 479-487.

LIU, L., ZHANG, L., XU, Y., GOTSMAN, C., AND GORTLER, S.J. A local/global approach to mesh parameterization. 2008. *Computer Graphics Forum* 27, 5, 1495-1504.

MARTIN, S., KAUFMANN, P., BOTSCH, M., WICKE, M., AND GROSS, M. Polyhedral finite elements using harmonic basis functions. 2008. *Computer Graphics Forum* 27, 5, 1521-1529.

SORKINE, O. AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. Symposium on Geometry Processing*, 109-116.

SORKINE, O., AND COHEN-OR, D. 2004. Least-squares meshes. In *Proc. of Shape Modeling International*, 191-199.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H. 2004. Laplacian surface editing. In *Proc. Symposium on Geometry Processing*, 175-184.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph.* 26, 3, 80.

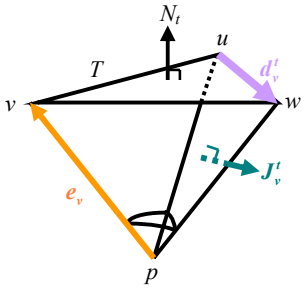
URAGO, M. 2000. Analytical integrals of fundamental solution of three-dimensional Laplace equation and their gradients. In *Trans. of the Japan Soc. of Mech. Eng.* 66, 642, 254-261.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Computer Graphics Forum* 26, 3, 265-274.

WEBER, O., BEN-CHEN, M., AND GOTSMAN, C. 2009. Complex barycentric coordinates with applications to planar shape deformation. *Computer Graphics Forum* 28, 2, 587-597.

Appendix A

The mappings φ and ψ. The mappings φ and ψ are defined on the vertices and faces of the mesh, respectively, so we must provide for each face a scalar value ψ_t and for each vertex a scalar value φ_v. The values of φ_v are determined as a sum of values on the faces neighboring v. Given a point $p \in \Omega$, and a face $t = (u,v,w) \in F$, we define a tetrahedron T spanned by these four points, as in Figure 15.



$$\tilde{e}_v = \frac{e_v}{\|e_v\|}$$

$$R_v^t = \|e_w\| + \|e_u\|$$

$$C_v^t = \frac{1}{4\pi\|d_v^t\|}\log\left(\frac{R_v^t + \|d_v^t\|}{R_v^t - \|d_v^t\|}\right)$$

$$\tilde{C}_v^t = \frac{1}{2\pi\left(R_v^t + \|d_v^t\|\right)\left(R_v^t - \|d_v^t\|\right)}$$

$$P_t = N_t \times \sum_{i \in t} d_i^t C_i^t - \omega_t N_t$$

Figure 15: *Notations for the definitions of $\varphi_v(p)$, $\psi_t(p)$ and their derivatives.*

On this tetrahedron, $4\pi\omega_t$ is the signed solid angle at the point p, subtended by the face t, and vol_t is its signed volume. N_t is the normalized outward pointing normal of t, and A_t is the area of the face t. Following Urago [2000] we obtain:

$$\psi_t = -\sum_{i \in t} C_i^t \left(J_i^t \cdot N_t\right) - \frac{3}{A_t}\omega_t vol_t$$

$$\varphi_v = \sum_{t \in N(v)} \frac{1}{2A_t} P_t \cdot J_v^t$$

The gradients. Again following Urago [2000], and taking the derivative of φ_v:

$$\nabla\psi_t = -P_t$$

$$\nabla\varphi_v = \sum_{t \in N(v)} \frac{1}{2A_t} P_t \times d_v^t$$

The Hessians. To derive the Hessian matrices for ψ_t and φ_v we need the Jacobian matrix of P_t, and the gradient vector of ω_t. These are:

$$\nabla\omega_t = \sum_{i=1}^{3} J_{v_i}^t \tilde{C}_{v_i}^t \left(\left\|e_{v_{i+1}}\right\|^{-1} + \left\|e_{v_{i+2}}\right\|^{-1}\right)$$

$$J\left(P_{t=(u,v,w)}\right) = \begin{pmatrix} (\tilde{e}_v + \tilde{e}_w)\tilde{C}_u^t \\ (\tilde{e}_u + \tilde{e}_w)\tilde{C}_v^t \\ (\tilde{e}_u + \tilde{e}_v)\tilde{C}_w^t \end{pmatrix}_{3x3}^T \begin{pmatrix} d_u^t \\ d_v^t \\ d_w^t \end{pmatrix}_{3x3} [N_t]_\times^T + \nabla\omega_t^T N_t$$

where, given a vector v , $[v]_\times$ is the skew symmetric matrix, such that for any vector w, $[v]_\times w = v \times w$. Finally the Hessian matrices of φ_v and ψ_t are:

$$H(\psi_t) = -J(P_t)$$

$$H(\varphi_v) = -\sum_{t \in N(v)} \frac{1}{2A_t}\left[d_v^t\right]_\times J(P_t)$$